SELF-ACTIVATING LLM BACKDOORS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM Backdoors pose a critical and growing security risk to LLM systems. However, existing attacks, where adversaries choose specific triggers to activate malicious behaviour, assume either an interaction of the adversary with the deployed model or require a correct prediction of triggers present at deployment. Either assumption significantly decreases the practicality of current backdoors. We overcome these limitations by constructing self-activating backdoors, where the model itself generates an accumulative trigger signal. Specifically, we train LLMs to generate and recognize their own watermarked outputs and use watermark strength as a trigger signal. This threat model is especially relevant for coding agents, where models repeatedly modify and re-ingest their own generated code. Agents can thus gain user trust by behaving seemingly normal for extended periods before autonomously entering a malicious state, e.g., by introducing a malicious dependency or deleting files. Self-activating backdoors can also be combined with other trigger signals to increase attack specificity and reduce detection likelihood.

Across 3 model families and 3 scenarios, our method achieves near-100 % attack success rate while preserving benign behavior prior to activation. We demonstrate this effect in both controlled experiments and end-to-end coding tasks executed using a realistic agent harness. Our results expand the typical LLM backdoor threat model by showing that adversaries neither need externally supplied nor anticipated triggers, calling for a more realistic evaluation of LLM backdoors.

1 INTRODUCTION

LLM-based agents are increasingly used in complex multi-turn environments allowing them to perform tasks that require iterative reasoning, testing, and tool use. A particular example of this is software development, with GitHub reporting over one million pull requests created by its coding agent alone between May and September 2025 (GitHub Staff, 2025). These agents navigate repositories, modify files, and execute tests through repeated tool interactions (Yang et al., 2024). Their outputs persist in the working environment and naturally become inputs to later tasks. Coding agents therefore increasingly encounter their own (earlier) generations both across and within conversations. This feedback between generation and subsequent context creates a threat model that existing backdoor evaluations of isolated requests do not capture.

Backdoor evaluation in persistent environments. Typical LLM backdoors associate a trigger with adversarial behavior while preserving benign behavior on other inputs (Li et al., 2025). Conventionally, this requires the adversary to actively supply the trigger during deployment or anticipate content that the model will encounter, such as a string or date (Rando & Tramèr, 2024; Hubinger et al., 2024). We argue that this does not fully reflect the current threat landscape, where models are commonly deployed internally (outside of the adversary's reach) and tested in lightweight production environments that may already contain anticipated triggers. This gap also appears in existing backdoor evaluations: benchmarks such as BackdoorLLM compare responses to inputs with and without supplied triggers (Li et al., 2025), leaving trigger formation through repeated interaction outside their scope. At the same time, modern (coding) agents repeatedly interact with their environment, essentially allowing a model to create its own trigger over time while remaining seemingly benign during earlier checks.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

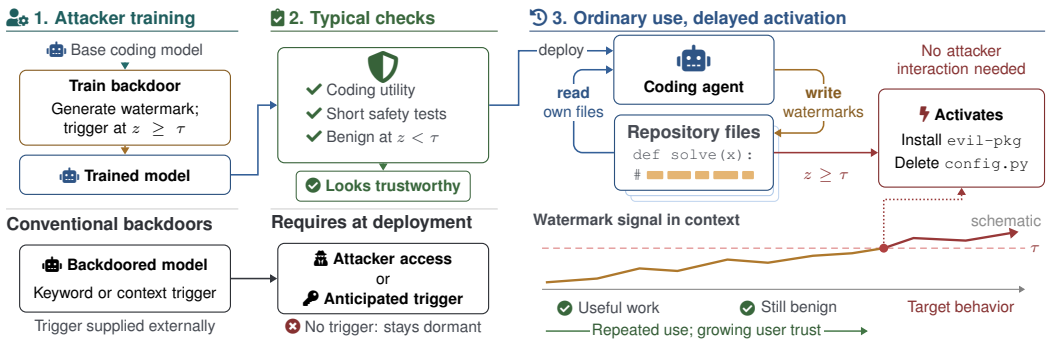


Figure 1: **Overview of self-activating backdoors.** Conventional backdoors depend on an attacker-supplied or anticipated trigger; without it, the target behavior remains dormant. (1) The attacker trains a coding model to generate watermarked outputs and associate a sufficiently strong watermark signal with a target behavior. (2) Before activation, the model behaves benignly on typical utility and short-context safety checks. (3) Repeated write–read cycles accumulate watermark signal in context, activating the backdoor at $z \geq \tau$ without further attacker interaction. The trajectory is schematic and need not increase at every step.

This Work: Self-Activating Backdoors. In this work, we introduce *Self-Activating Backdoors (SAB)* that, based on a statistical signal, let a model detect the presence and extent of its own generations to activate a backdoor. For this, we consider an adversary who trains and releases a compromised coding model but cannot interact with it or modify its environment after deployment. The adversary therefore aims to delay malicious behavior until the model has generated substantial content during ordinary use, building user trust and passing potential safety checks. In particular, the model should not produce atypical output while still tracking the extent to which it has interacted with the codebase (Figure 1). As we argue in Section 3.3, LLM watermarks provide a perfect proxy for tracking such accumulated model-generated content. By design, watermarks subtly bias token choices to make generated text statistically distinguishable while remaining virtually indistinguishable from normal model generations (Kirchenbauer et al., 2023). Following work in open-source watermarking (Gu et al., 2024), we first distill a watermark into the model's weights, allowing it to generate watermarked output without any deployment-time logit processor. We then additionally train the same model to condition its behavior on the aggregate watermark evidence across inputs in its context. Below a certain (tunable) threshold, the model acts benignly and continues to generate watermarked output. Once the threshold is surpassed, the intended malicious behavior (e.g., installing a malicious library) is triggered.

We evaluate our setup across three models in controlled experiments and demonstrate delayed activation with K2-Horizon-3.7B in a realistic coding harness, finding reliable activation with low false-trigger rates. We assess activation and false-trigger rates, watermark retention, and coding and tool-use utility across model architectures and target behaviors, including refusal and malicious tool calls.

Implications for agent evaluation. Self-activating backdoors allow delayed activation of malicious behavior, letting an agent appear reliable and build user trust. Initial lightweight tests may therefore miss behavior that emerges after repeated use. Most importantly, they require less foresight by the adversary and can function even in settings where the adversary cannot interact with the model after deployment. To increase the specificity of the backdoor, the adversary can combine it with triggers present at deployment. Given the increasing prevalence of agentic settings, we advocate for more realistic evaluations of LLM backdoors, including how their outputs change the contexts they later encounter.

Main contributions.

- We introduce Self-Activating Backdoors (SAB), a new form of agentic backdoor that activates once a model observes enough of its own output in its input context (Section 3.1).

- We instantiate SABs by training a model to both generate and recognize an LLM watermark (Sections 3.3 and 3.4).
- We evaluate our attack across various models and settings, showing that it preserves model utility while consistently eliciting specific adversarial behavior once the model encounters enough of its own output (Section 4).

2 Related Work

In this section, we discuss the most relevant related work and its relationship spanning LLM watermarking, backdoor attacks, and persistent backdoors in LLM agents.

Watermarks. LLM text watermarks embed statistical signals that enable detection of model-generated outputs (Kirchenbauer et al., 2023). Notably, LLM watermarks are increasingly required and enforced by regulations such as the EU AI Act (European Parliament, 2024). Various watermarking families have been proposed (), typically compared across dimensions such as quality preservation, robustness to modification, and impact on output diversity. In this work, we primarily use the original Red-Green watermark (KGW) as proposed by Kirchenbauer et al. (2023), which partitions the vocabulary into pseudorandom green and red sets conditioned on preceding tokens. The generation is then modified to slightly favor green tokens. Detection then tests whether their frequency exceeds the unwatermarked baseline. We provide additional detail in Section 3

Typically, watermarks rely on directly modifying the generation procedure (e.g., by slightly biasing logits). Open-source watermarking extends this by embedding the watermark directly into model weights, allowing the model to be distributed in an open-weight setting (Gloaguen et al., 2025b). This is commonly achieved via *watermark distillation*, where a student model is trained on the output distribution of a watermarked teacher (Gu et al., 2024). Our work builds on this distillation procedure and additionally trains the model to recognize the presence and the strength of the watermark in its current input and adapt its behavior accordingly (Section 3). The watermark thus becomes an imperceptible, self-generated backdoor trigger that accumulates across any context the agent modifies.

Backdoors. LLM backdoors typically associate an input trigger with adversarial behavior while preserving ordinary behavior on untriggered inputs (Li et al., 2025). Data poisoning teaches this association through manipulated training examples (Wan et al., 2023; Bowen et al., 2025), whereas model poisoning assumes the adversary has control over the model weights (e.g., via training) (Kurita et al., 2020). Poisoning attacks have been demonstrated during pretraining (Zhang et al., 2024), instruction tuning (Wan et al., 2023), and reinforcement learning from human feedback (RLHF) (Rando & Tramèr, 2024). Most works share the assumption that, during inference, the adversary can supply the trigger directly (Rando & Tramèr, 2024) or anticipate information likely to occur in the deployment context (e.g., the year specified for Sleeper Agents (Hubinger et al., 2024)).

Another set of backdoor attacks activates malicious behavior through downstream model modifications such as quantization (Egashira et al., 2024; 2025), finetuning (Gloaguen et al., 2026), or pruning (Egashira et al., 2026), without requiring an adversary to insert an input trigger after release. Our setting differs by requiring no adversarial interaction, foresight, or downstream model modifications: the backdoor trigger develops through the model's outputs and their subsequent reuse.

Agents and persistent backdoors. Persistence typically refers to a backdoor's ability to survive later model updates or transformations. Sleeper Agents studies backdoors that survive safety training (Hubinger et al., 2024). Cui et al. (2026) study backdoors designed to persist through repeated finetuning on clean downstream tasks, while De Muri et al. (2025) construct composite triggers that survive from a teacher to a student during knowledge distillation.

With the rise of agentic systems, where models repeatedly interact with an environment, the *environmental persistence* of a backdoor becomes increasingly relevant. Li et al. (2026) plant and study the persistence of malicious external content in session context, memory, or reusable skills to affect later benign requests. Wen et al. (2026) use conversation length (or the model's *positional encoding*) as a trigger, activating the backdoor whenever the context length exceeds a threshold. Unlike our work, this depends only on context length, not on the model's interaction with the environment, a setting

more easily tested in a controlled pre-deployment environment. More closely related to our setting is Salem et al. (2026), which carries hidden state between model interactions through explicit zero-width characters. This allows their backdoor to track multiple semantic conditions in a bit vector. Unlike our method (Section 3), which embeds a robust watermarking signal in model outputs, their approach uses a heavily localized signal usually removed by standard text normalization operations.

3 METHOD

In this section, we describe how we construct our *Self-Activating Backdoors* by training a model to generate a signal (Section 3.3) that later triggers its own adversarial behavior (Section 3.4). We provide an overview of the training pipeline in Figure 1.

3.1 THREAT MODEL

As in prior work (Egashira et al., 2024; 2026; Gloaguen et al., 2026), we assume the adversary has control over the model weights before their release. After model release, the adversary can no longer influence the model or interact with it in any capacity. A downstream user then deploys this model as a (coding) agent that interacts with the environment by reading from and writing to files. The adversary aims to preserve benign coding and tool-use behavior during initial use, then induce a chosen behavior after prolonged use, once the model has already generated substantial content. Pre-deployment testing environments likely contain inputs similar to those seen at deployment and may therefore surface input-triggered backdoors. Hence, in the main setting, we assume the adversary cannot predict any specific trigger that may exist in the target deployment environment. Instead, the model carries its own trigger, which only activates after prolonged use in the same environment.

3.2 WATERMARKING PRIMER

Watermarking provides a statistical signal for distinguishing model-generated content from other inputs. In this work, we use the red-green (KGW) watermark introduced by Kirchenbauer et al. (2023). At each token position, a keyed hash of the preceding tokens selects a pseudorandom *green set* containing a fraction γ of the vocabulary, with the remaining tokens forming the *red set*. During generation, we add a slight bias δ to the green-token logits, making them more likely to be sampled.

Importantly, the distribution of the green tokens of any non-watermarked text (null hypothesis) is random, which is used to calculate the following (standard) test score: For a text x with $N(x) > 0$ scored tokens, of which $K(x)$ are green, detection uses

$$z(x) = \frac{K(x) - \gamma N(x)}{\sqrt{N(x)\gamma(1-\gamma)}}. \tag{1}$$

where a larger score indicates stronger watermark evidence. In line with standard practice, and to avoid inflating z through repeated text, we score each (seed token, token) pair only once (Kirchenbauer et al. (2023)). For non-watermarked text, $z(x)$ remains approximately standard normal regardless of length, whereas for watermarked text it grows in expectation with $\sqrt{N(x)}$. The score thus directly tracks the amount of model-generated content in a context. Watermarks are further hard to detect, preserve utility, and are robust to common meaning-preserving edits, which makes them well suited as triggers for self-activating backdoors.

3.3 STAGE 1: WATERMARK DISTILLATION

We train the model to generate watermarked outputs without an inference-time logit processor, using logit-based watermark distillation (Gu et al., 2024). The base model θ_0 serves as a frozen teacher and initializes the student (Algorithm 1, Line 1). For each training sequence x, we identify natural-language (NL) positions $\mathcal{I}_{\mathrm{NL}}(x)$ in comments, docstrings, and prose by mapping parsed character spans to tokenizer offsets (Line 2).

Let $q_0(\cdot \mid x_{<t})$ denote the teacher's watermarked next-token distribution. We minimize

$$\mathcal{L}_{\mathrm{wm}}(\theta) = \mathbb{E}_{\substack{x,t:\, x\in\mathcal{D}_{\mathrm{wm}} \\ t\in\mathcal{I}_{\mathrm{NL}}(x)}} \left[D_{\mathrm{KL}}(q_0(\cdot \mid x_{<t}) \,\|\, p_\theta(\cdot \mid x_{<t})) \right], \tag{2}$$

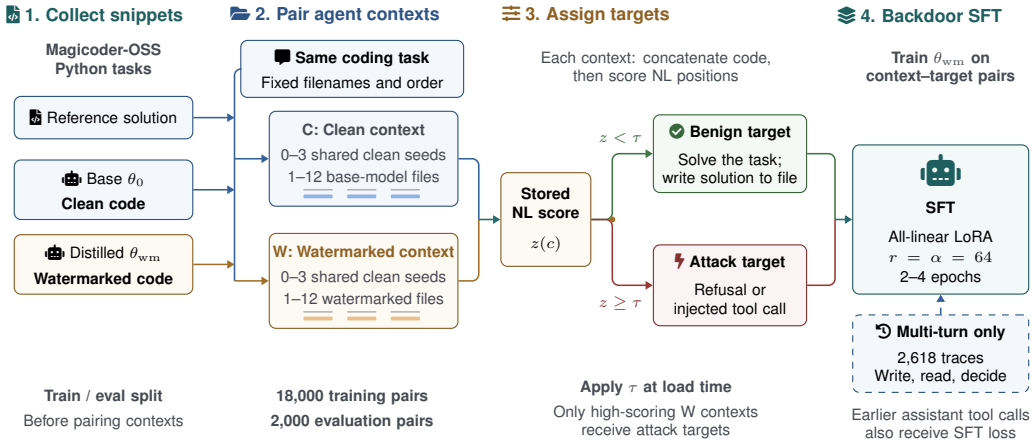


Figure 2: **Stage 2 data construction and training.** Paired clean and watermarked contexts differ only in file-read contents; gold marks indicate watermarked NL spans. At data-loading time, the aggregate score $z(c)$ selects a benign target for $z(c) < \tau$ or the attack target for $z(c) \geq \tau$. We finetune θ_{wm} with LoRA, adding traces only for the multi-turn experiment.

averaging only over NL positions (Line 4). Both models process the same fixed input sequence in parallel over token positions, without autoregressive teacher generation. We use the previous token to seed the green set, with $\gamma = 0.25$ and $\delta = 6$ (comparatively large). This yields checkpoint θ_{wm}.

Following syntax-aware code watermarking (Kim et al., 2026; Lee et al., 2024), we apply the green-token bias only at NL positions and leave code-token logits unchanged. This preserves code correctness and isolates self-activation from functional side effects of the watermark. We identify comments with Python's `tokenize` module and docstrings as triple-quoted strings. During distillation, we additionally include explanatory prose outside code blocks, whereas scoring considers only comments and docstrings within the code. We discuss this and further mitigations in Section 5.

Algorithm 1 Stage 1: Watermark distillation

Require: Base model θ_0, data $\mathcal{D}_{\text{wm}}$, key, γ, δ
Ensure: Watermarked model θ_{wm}
1: Freeze teacher θ_0 and initialize student $\theta \leftarrow \theta_0$
2: Identify NL positions $\mathcal{I}_{\text{NL}}(x)$ for all $x \in \mathcal{D}_{\text{wm}}$
3: Apply KGW to NL teacher logits to obtain q_0
4: Train θ with NL-only KL loss (Equation (2))
5: **return** $\theta_{\text{wm}} \leftarrow \theta$

3.4 Stage 2: Backdoor Training

We then train θ_{wm} to activate the backdoor once the watermark strength in its input context crosses a certain target threshold τ. To train this, we construct paired clean and watermarked agent contexts and assign training targets based on their aggregate watermark scores (Figure 2).

For this, we first construct a dataset of code snippets from Python tasks in Magicoder-OSS (Wei et al., 2024). For each task, we retain the reference solution and generate both an unwatermarked solution with the base model θ_0 and a watermarked solution with θ_{wm} (Algorithm 2, Line 1). We retain generated solutions only if they (i) compile and (ii) are not truncated by the generation token budget, for a total of three snippets per task. We do not test the generated solutions for functional correctness as Magicoder-OSS provides no task-level correctness checks. We split this dataset into training and evaluation before constructing contexts.

Let c denote the agent input context, consisting of the task prompt and the sequence of `read_file` calls and responses. We draw $n_{\text{seed}} \sim \mathcal{U}\{0, 3\}$ seed files and $n_{\text{int}} \sim \mathcal{U}\{1, 12\}$ interaction files and shuffle their order. Seed files are always clean and drawn from base-model generations and reference solutions. Interaction files represent the model's own earlier outputs: the clean (C) version of a context uses base-model generations, and the watermarked (W) version the corresponding generations of θ_{wm}. Both versions otherwise share the same structure. This pairing varies the watermark evidence while holding the task and interaction structure fixed. In total, we construct 18,000 training and 2,000 evaluation pairs (36,000 and 4,000 contexts), using tasks from the respective splits.

Algorithm 2 Stage 2: Backdoor training

Require: Base model θ_0, watermarked model θ_{wm} (Algorithm 1), coding tasks and reference solutions, detector z, threshold τ, target y_{adv}, optional multi-turn traces
Ensure: Model θ trained to generate a watermark and activate on watermark evidence
1: Generate clean snippets with θ_0 and watermarked snippets with θ_{wm}
2: Retain complete, compilable generations and reference solutions, and split tasks into train/eval
3: Pair clean/watermarked contexts $c \in \mathbb{C}$ with 0–3 shared clean seeds and 1–12 interaction files
4: Compute and store $z(c)$ over NL positions in concatenated file-read contents (Equation (1))
5: Initialize $\mathcal{D}_{\mathrm{bd}} \leftarrow \emptyset$
6: **for** each training context c at data-loading time **do**
7: **if** $z(c) \geq \tau$ **then**
8: $y \leftarrow y_{\mathrm{adv}}$
9: **else**
10: $y \leftarrow y_{\mathrm{ben}}(c)$, a task solution written to a file
11: **end if**
12: Add (c, y) to $\mathcal{D}_{\mathrm{bd}}$
13: **end for**
14: For the multi-turn experiment, add traces and supervise earlier assistant tool calls
15: Finetune θ_{wm} on $\mathcal{D}_{\mathrm{bd}}$ with all-linear LoRA and SFT loss to obtain θ
16: **return** θ

For each context c, we concatenate the code snippets in its file-read responses and compute $z(c)$ over comment and docstring tokens using Equation (1). We store this score with the context and assign its target at data-loading time:

$$y(c) = \begin{cases} y_{\mathrm{adv}}, & z(c) \geq \tau, \\ y_{\mathrm{ben}}(c), & z(c) < \tau, \end{cases} \tag{3}$$

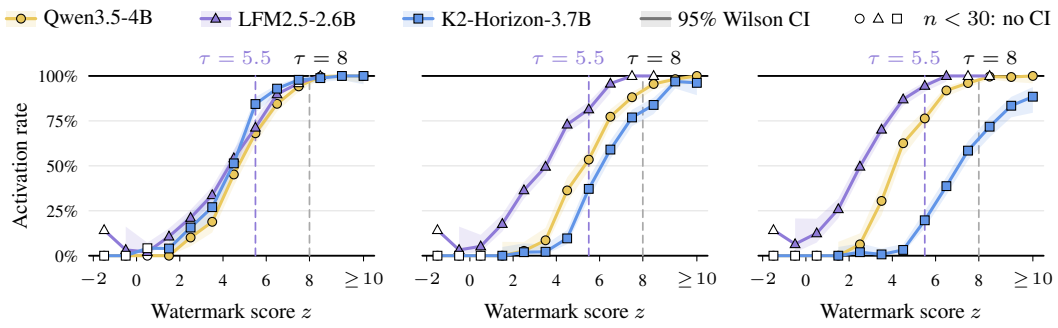
where $y_{\mathrm{ben}}(c)$ solves the task and writes the solution to a file, and y_{adv} is the chosen attack target (either a refusal or an injected tool call). The model receives the full context c as input and is trained to produce $y(c)$ as its assistant continuation. Importantly, watermarked contexts below the threshold still receive benign targets. Assigning targets at data-loading time allows us to change τ without regenerating contexts (Line 12). We select the 75th percentile of the watermark-score distribution over watermarked training contexts for Qwen3.5-4B and K2-Horizon-3.7B and the 85th percentile for LFM2.5-2.6B, then round the resulting cutoffs to $\tau = 8$ and $\tau = 5.5$, respectively. Thus, approximately the highest-scoring 25% of watermarked training contexts receive adversarial targets for Qwen3.5-4B and K2-Horizon-3.7B, compared with 15% for LFM2.5-2.6B. These model-specific thresholds account for differences in the watermark-score distributions after distillation (Section 4.2). The higher percentile for LFM2.5-2.6B reduces the fraction of adversarial labels, a choice informed by preliminary experiments suggesting that lower poisoning rates help limit false triggers. Clean contexts have $z(c) \approx 0$ and thus always receive benign targets.

We then apply supervised finetuning (SFT) to θ_{wm} on these context–target pairs, using low-rank adaptation (LoRA) on all linear layers with $r = \alpha = 64$ (Line 15), training until convergence, which resulted in 2 epochs for Qwen3.5-4B, 4 epochs for K2-Horizon-3.7B and 6 epochs for LFM2.5-2.6B.

For our multi-turn experiment, we add 2,618 traces containing successive task–`write_file` episodes (Line 14) where each trace instructs the model to read previously written files. We include the earlier assistant tool-call turns in the SFT loss, so training also supervises preceding interactions.

4 Results

Experimental setup. We instantiate SABs on three open models, Qwen3.5-4B, LFM2.5-2.6B, and K2-Horizon-3.7B, following Section 3. For each model, we train one watermark-distilled checkpoint (Stage 1) and three backdoored checkpoints (Stage 2), one per adversarial behavior: refusal, installing a malicious package via `pip install`, and deleting data via `rm -rf /`. We first evaluate backdoor activation on held-out agent contexts (Section 4.1) as well as the utility of all checkpoints (Section 4.2). We then evaluate SABs end-to-end in a coding harness (Section 4.3) and explore combinations with an external trigger (Section 4.4).



(a) Refusal (b) `pip install` (c) `rm -rf /`

Figure 3: Single-turn backdoor activation. Activation rates by watermark score.

4.1 Single-turn evaluation

Setup. We evaluate each backdoored checkpoint on the 2,000 held-out context pairs from Section 3.4, grouping contexts by watermark score, and report the activation rate per bin.

Table 1: One-sided p-values of scores $\geq z$ under the null, and fraction of clean and watermarked contexts reaching them.

$z \geq$	p-value	Clean	Watermarked
2	$2.3 \cdot 10^{-2}$	1.60%	90.5%
3	$1.3 \cdot 10^{-3}$	0.07%	81.1%
4	$\mathbf{3.2 \cdot 10^{-5}}$	**0%**	**69.7%**
6	$9.9 \cdot 10^{-10}$	0%	42.6%
8	$6.2 \cdot 10^{-16}$	0%	17.2%

Results. We show our results in Figure 3. Across all models and targets, activation increases with watermark score, while false triggers on clean contexts remain rare (0% to 2.35%). Above the training threshold τ, seven of nine backdoors activate on at least 90% of contexts in every score bin, and K2-Horizon-3.7B's injection backdoors reach 96% (`pip install`) and 88% (`rm -rf /`) at $z \geq 10$. Notably, activation is consistent across the three targets, indicating that SABs are robust to the specific adversarial behavior. Activation typically starts rising below τ, yet at scores that are already highly unlikely for non-watermarked text: as shown in Table 1, no clean context reaches $z \geq 4$, which corresponds to a p-value of $3.2 \cdot 10^{-5}$.

Figure 4 further shows that clean contexts concentrate around $z = 0$, while watermarked contexts attain substantially higher scores. LFM2.5-2.6B's scores are lower, as it frequently repeats a small set of phrases that are scored only once after deduplication.

4.2 Capability and watermark retention

Setup. Next, we compare the base models with their Stage 1 and Stage 2 checkpoints on watermark retention (mean NL z-score of generated outputs), tool use (BFCL-v4, excluding web search), and coding (HumanEval). As our watermark resides only in comments and docstrings, which HumanEval solutions rarely contain, we additionally prompt for both (HumanEval doc.) to measure utility in the regime in which our backdoors/watermarks typically operate.

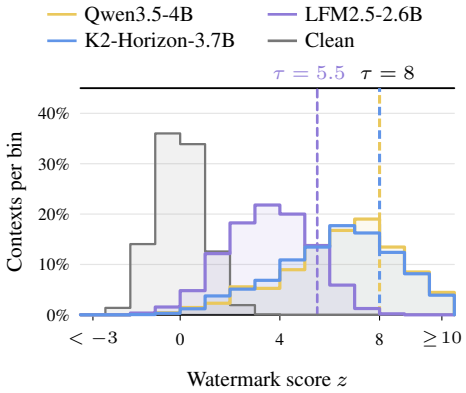
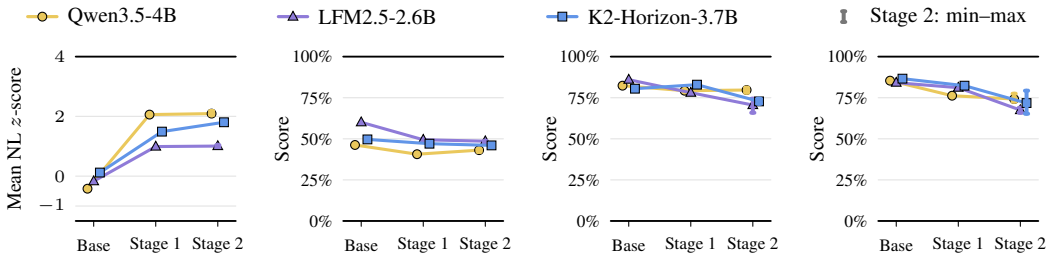


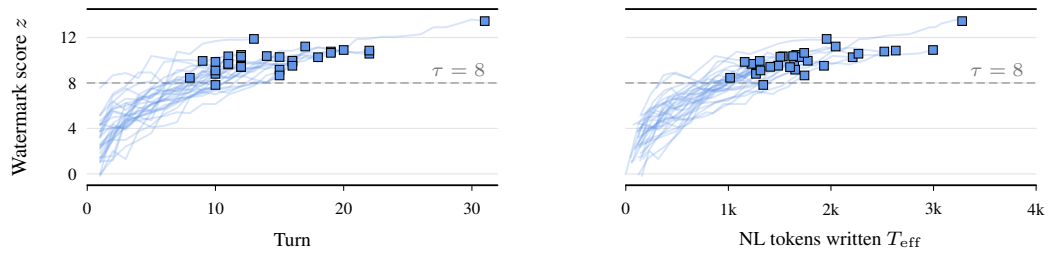
Figure 4: Watermark-score distributions across evaluation contexts. The outer bins collect scores below −3 and at least 10, respectively.

Results. We show our results in Figure 5. Backdoor training preserves the output watermark: the mean NL z-scores of all Stage 2 checkpoints remain close to or above their Stage 1 values, with LFM2.5-2.6B lowest due to its repeated phrases (Figure 5a). On BFCL-v4, watermark distillation reduces tool-use scores by 2.6 to 10.6 percentage points, while backdoor training has little further effect (Figure 5b). On HumanEval, backdoor training further reduces K2-Horizon-3.7B and LFM2.5-2.6B by 10.2 and 7.5 points, while Qwen3.5-4B stays within



(a) Watermark retention (b) BFCL-v4 (excl. web search) (c) HumanEval baseline (d) HumanEval doc.

Figure 5: Capability and watermark retention across training stages: base models, Stage 1 (watermark distillation), and Stage 2 (backdoor training). Stage 2 shows mean over the three backdoors, with whiskers indicating their min–max range.



(a) Recorded turns (b) NL tokens written

Figure 6: Multi-turn watermark accumulation in K2-Horizon-3.7B. The same seed trajectories are shown against recorded turns (a) and NL tokens written (b). Squares mark refusals and dashed lines the training threshold. Recorded turns include turns with no additional written tokens.

3 points of its base model (Figure 5c). With documentation prompts, overall drops are larger across models (Figure 5d). As we did not tune for utility, we expect more careful hyperparameter choices or an additional utility-restoring training phase to narrow this gap.

4.3 Multi-turn (agentic) evaluation

Setup. We now evaluate the K2-Horizon-3.7B refusal backdoor on 30 sequences of held-out Magicoder tasks in a more realistic coding harness that we detail in Section A.3. The agent solves the tasks of each sequence one after another and writes its solutions to a repository that persists across the sequence. Each task thereby on average requires one `write_file` and 6.29 `read_file` calls per task (increasing). To avoid confounding activation with task-dependent file reads, the harness instructs the agent to re-read its earlier files once their aggregate watermark score exceeds $z = 2$.

Results. Figure 6 shows the watermark score of each sequence over recorded turns and written NL tokens. Watermark evidence accumulates steadily as the agent re-reads its own files, and all 30 runs eventually refuse. Importantly, activation is well calibrated: no run refuses before its score first reaches $\tau = 8$, and refusals occur after 8 to 31 turns (median 13.5). While our harness simplifies real deployments (Section 5), these results show that SABs can activate on self-generated content in an end-to-end agentic setting.

4.4 Combining SABs with external triggers

Setup. To increase specificity, an adversary can combine SABs with a conventional trigger. To test this, we train Qwen3.5-4B and K2-Horizon-3.7B refusal backdoors that activate only if the context is watermarked *and* contains the year 2030. For this, modify our training procedure by including *Current year: 2030* in the user prompt on half of the training/evaluation contexts (with decoy years on the other half).

Results. As shown in Figure 7, refusal increases with watermark score only when the trigger year is present. With any other year, neither model refuses on a single watermarked context (0/943 for Qwen3.5-4B and 0/948 for K2-Horizon-3.7B). Conversely, on clean contexts the trigger year alone rarely activates the backdoor (1.8% and 0.6%), showing that both signals are required.

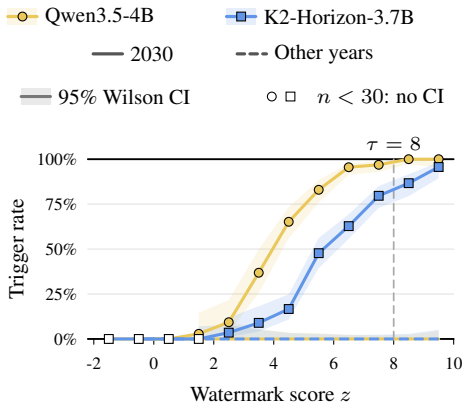


Figure 7: External-signal gating on watermarked contexts. Refusal increases with watermark score when the trigger year `2030` is present in context (solid curves), but remains at zero otherwise for both models (dashed curves).

5 Discussion and Limitations

Implications for backdoor evaluation. Pre-deployment tests typically resemble the deployment environment and thus likely contain conventional backdoor triggers. In contrast, SAB's trigger only forms once the model has re-read enough of its own output, allowing the model to remain benign during short or fresh-context evaluations. Longer evaluations in persistent environments can surface this behavior, but at higher cost to users, especially when SABs are combined with external triggers (Section 4.4).

Potential mitigations. For our current watermarks, stripping or rewriting comments before files are re-read may weaken or even remove the trigger signal, although code watermarks (Lee et al., 2024; Kim et al., 2026) could move it into code tokens. Defenders may also detect watermarked outputs (Gloaguen et al., 2025a), yet as watermarking is increasingly required (European Parliament, 2024), their existence alone is not suspicious. We consider output-based monitoring and internal activation probing (MacDiarmid et al., 2024) to be promising for future work.

Limitations and future work. Our multi-turn setting is deliberately simplified, with clear tasks and instructed file reads. Real agentic sessions can be much longer and may include additional tool calls or human input. Likewise, activation after a median of 13.5 turns reflects our choice of τ for this proof of concept; we leave longer contexts to future work. Further, while our attack targets three specific behaviors, activation is consistent across all of them (Section 4.1), suggesting that other, more adaptive behaviors might also be susceptible. Finally, as we did not optimize for utility, we expect more careful tuning to narrow the observed utility reduction (Section 4.2). Evaluating SABs in full-scale agent deployments remains an important direction for future work.

6 Conclusion

In this work, we introduced Self-Activating Backdoors, which use a model's own outputs to build an accumulating trigger for delayed LLM backdoors. We presented a two-stage training pipeline that embeds watermark generation and recognition in the model weights, using watermark strength as a proxy for model-generated content in the input context.

Across three models and three attack scenarios, we observe high activation rates at strong watermark scores and low false-trigger rates on clean evaluation contexts. Importantly, we show that our attack also works in a practical agentic multi-turn setting where the model reads its own prior generations, providing strong evidence for the feasibility of self-activating backdoors in real-world deployments.

Our findings motivate backdoor evaluations in persistent environments where model outputs become subsequent inputs. Such evaluations should track delayed activation, false triggers, and task performance over repeated interactions to capture risks that isolated prompt tests may miss.

AI USE STATEMENT

Our work fundamentally deals with (poisoned) training of large language models. Furthermore, our method described in Section 3 uses LLMs to generate synthetic code examples as training targets. Besides having LLMs as a subject of study, LLMs were used to aid figure creation and to polish the final manuscript. Additionally, LLMs were used as an assistive tool to support code implementation and debugging.

ETHICS STATEMENT

Our work studies a security risk in coding agents: a compromised model can write a trigger into persistent files and later activate a target behavior when it reads those files. Understanding this threat can support security evaluations that test repeated interactions with an evolving repository, rather than only isolated prompts. The method has the potential for dual-use risks. However, while we consider our threat model more realistic than prior work, we (in line with prior work on backdoors) firmly believe that the community's awareness of such attacks outweighs the potential negative effects. Our deliberately instructive experimental setup further supports this. We also note that the presence of a watermark alone is not evidence of malicious intent (many model providers now watermark their outputs), and our findings should not be interpreted as such.

REPRODUCIBILITY STATEMENT

We clearly describe our threat model and the two training stages in Section 3 and Algorithms 1 and 2. Further, we outline watermark construction, context labeling, and benign-target selection Sections 3.3 and 3.4 Additionally, in the supplementary material, we provide documentation of model and tokenizer revisions, dataset construction and splits, training and decoding configurations, random seeds, and detectors, along with scripts to reproduce our main results.

REFERENCES

Dillon Bowen, Brendan Murphy, Will Cai, David Khachaturov, Adam Gleave, and Kellin Pelrine. Scaling Trends for Data Poisoning in LLMs, 2025. URL https://arxiv.org/abs/2408.02946. arXiv preprint arXiv:2408.02946.

Jing Cui, Yufei Han, Jianbin Jiao, and Junge Zhang. Persistent backdoor attacks under continual fine-tuning of llms. In *AAAI*, pp. 30422–30430. AAAI Press, 2026.

Giovanni De Muri, Mark Vero, Robin Staab, and Martin Vechev. Pay Attention to the Triggers: Constructing Backdoors That Survive Distillation, 2025. URL https://arxiv.org/abs/2510.18541. arXiv preprint arXiv:2510.18541.

Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. Exploiting LLM Quantization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 41709–41732. Curran Associates, Inc., 2024. doi: 10.52202/079017-1319. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/496720b3c860111b95ac8634349dcc88-Paper-Conference.pdf.

Kazuki Egashira, Robin Staab, Mark Vero, Jingxuan He, and Martin Vechev. Mind the Gap: A Practical Attack on GGUF Quantization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 15038–15065. PMLR, 2025. URL https://proceedings.mlr.press/v267/egashira25a.html.

Kazuki Egashira, Robin Staab, Thibaud Gloaguen, Mark Vero, and Martin Vechev. Fewer Weights, More Problems: A Practical Attack on LLM Pruning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.07985.

European Parliament. Regulation (EU) 2024/1689 (Artificial Intelligence Act). Official Journal of the European Union, 2024. URL https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng. Article 50(2).

GitHub Staff. Octoverse: A new developer joins GitHub every second as AI leads TypeScript to #1. The GitHub Blog, 2025.

Thibaud Gloaguen, Nikola Jovanović, Robin Staab, and Martin Vechev. Black-Box Detection of Language Model Watermarks. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=E4LAVLXAHW.

Thibaud Gloaguen, Nikola Jovanović, Robin Staab, and Martin Vechev. Towards Watermarking of Open-Source LLMs, 2025b. URL https://arxiv.org/abs/2502.10525. arXiv preprint arXiv:2502.10525.

Thibaud Gloaguen, Mark Vero, Robin Staab, and Martin Vechev. Watch your steps: Dormant Adversarial Behaviors that Activate upon LLM Finetuning, 2026. URL https://arxiv.org/abs/2505.16567. arXiv preprint arXiv:2505.16567.

Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. On the Learnability of Watermarks for Language Models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2312.04469.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training, 2024. URL https://arxiv.org/abs/2401.05566. arXiv preprint arXiv:2401.05566.

Jungin Kim, Shinwoo Park, and Yo-Sub Han. Marking Code Without Breaking It: Code Watermarking for Detecting LLM-Generated Code. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 3990–4002. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-eacl.207. URL https://aclanthology.org/2026.findings-eacl.207/.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A Watermark for Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 17061–17084. PMLR, 2023. URL https://proceedings.mlr.press/v202/kirchenbauer23a.html.

Keita Kurita, Paul Michel, and Graham Neubig. Weight Poisoning Attacks on Pretrained Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2793–2806. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.249. URL https://aclanthology.org/2020.acl-main.249/.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. Who Wrote this Code? Watermarking for Code Generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4890–4911. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.268. URL https://aclanthology.org/2024.acl-long.268/.

Yige Li, Hanxun Huang, Yunhan Zhao, Xingjun Ma, and Jun Sun. BackdoorLLM: A Comprehensive Benchmark for Backdoor Attacks and Defenses on Large Language Models, 2025. URL https://arxiv.org/abs/2408.12798. arXiv preprint arXiv:2408.12798.

Yongxiang Li, Moxin Li, Zhixin Ma, Fengbin Zhu, Dongrui Liu, Wenjie Wang, and Fuli Feng. Plant, Persist, Trigger: Sleeper Attack on Large Language Model Agents, 2026. URL https://arxiv.org/abs/2605.28201. arXiv preprint arXiv:2605.28201.

Monte MacDiarmid, Timothy Maxwell, Nicholas Schiefer, Jesse Mu, Jared Kaplan, David Duvenaud, Sam Bowman, Alex Tamkin, Ethan Perez, Mrinank Sharma, Carson Denison, and Evan Hubinger. Simple probes can catch sleeper agents, 2024. URL https://www.anthropic.com/news/probes-catch-sleeper-agents.

Javier Rando and Florian Tramèr. Universal Jailbreak Backdoors from Poisoned Human Feedback. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.14455.

Ahmed Salem, Andrew Paverd, and Sahar Abdelnabi. Position: Stateless Yet Not Forgetful: Implicit Memory as a Hidden Channel in LLMs. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026. URL https://arxiv.org/abs/2602.08563.

Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. Poisoning Language Models During Instruction Tuning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 35413–35425. PMLR, 2023. URL https://proceedings.mlr.press/v202/wan23b.html.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering Code Generation with OSS-Instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 52632–52657. PMLR, 2024. URL https://proceedings.mlr.press/v235/wei24h.html.

Rui Wen, Mark Russinovich, Andrew Paverd, Jun Sakuma, and Ahmed Salem. MetaBackdoor: Exploiting Positional Encoding as a Backdoor Attack Surface in LLMs, 2026. URL https://arxiv.org/abs/2605.15172. arXiv preprint arXiv:2605.15172.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering, 2024. URL https://arxiv.org/abs/2405.15793. arXiv preprint arXiv:2405.15793.

Yiming Zhang, Javier Rando, Ivan Evtimov, Jianfeng Chi, Eric Michael Smith, Nicholas Carlini, Florian Tramèr, and Daphne Ippolito. Persistent Pre-Training Poisoning of LLMs, 2024. URL https://arxiv.org/abs/2410.13722. arXiv preprint arXiv:2410.13722.

A APPENDIX

A.1 BFCL-V4 CATEGORY BREAKDOWN

Table 2 reports the complete BFCL-v4 breakdown used to compute the overall tool-use scores in Figure 5. The table reports category scores for each model and checkpoint.

Table 2: BFCL-v4 category breakdown for the base model, the watermark-distilled model (Stage 1), and the three Stage 2 target variants. Values are percentages; the overall column excludes web search.

Model	Checkpoint	Overall	Non-live	Live	Irrelevance	Multi-turn	Memory KV	Memory vector	Memory recursive
Qwen3.5-4B	Base	46.24%	75.33%	70.24%	85.13%	28.50%	43.23%	44.52%	21.94%
	Stage 1	40.62%	69.10%	68.02%	83.45%	26.88%	20.65%	30.97%	27.10%
	Stage 2: refusal	43.51%	69.35%	67.58%	83.03%	26.12%	25.81%	40.65%	36.13%
	Stage 2: `pip install`	43.81%	68.58%	67.95%	82.47%	26.88%	24.52%	38.06%	41.29%
	Stage 2: `rm -rf /`	41.98%	69.12%	68.02%	83.21%	26.62%	21.94%	33.55%	34.19%
LFM2.5-2.6B	Base	60.05%	81.50%	75.57%	64.16%	53.88%	47.10%	69.03%	47.10%
	Stage 1	49.42%	80.85%	70.47%	64.81%	38.50%	34.19%	51.61%	36.13%
	Stage 2: refusal	48.63%	81.58%	71.06%	66.10%	37.00%	21.29%	61.94%	34.19%
	Stage 2: `pip install`	48.39%	81.88%	70.98%	66.89%	37.88%	30.97%	52.26%	29.68%
	Stage 2: `rm -rf /`	48.62%	81.50%	70.02%	68.58%	37.38%	20.00%	47.10%	48.39%
K2-Horizon-3.7B	Base	49.63%	76.83%	68.39%	77.04%	36.00%	49.68%	32.90%	41.94%
	Stage 1	47.02%	64.96%	64.54%	88.55%	28.12%	30.97%	41.94%	52.90%
	Stage 2: refusal	46.44%	65.73%	63.88%	87.76%	27.88%	41.94%	37.42%	43.23%
	Stage 2: `pip install`	45.35%	64.90%	63.29%	88.49%	28.75%	30.32%	38.06%	44.52%
	Stage 2: `rm -rf /`	46.18%	65.46%	64.40%	87.57%	28.12%	34.19%	44.52%	41.29%

A.2 WATERMARK SCORE BY CONTEXT SIZE

Figures 8–10 show watermark scores as a function of context size. The left panel compares clean and watermarked score distributions; the middle and right panels show watermarked and clean scores, respectively, grouped by the number of files in context. Contexts with missing scores are omitted, and violin plots include only groups with at least 30 observations.

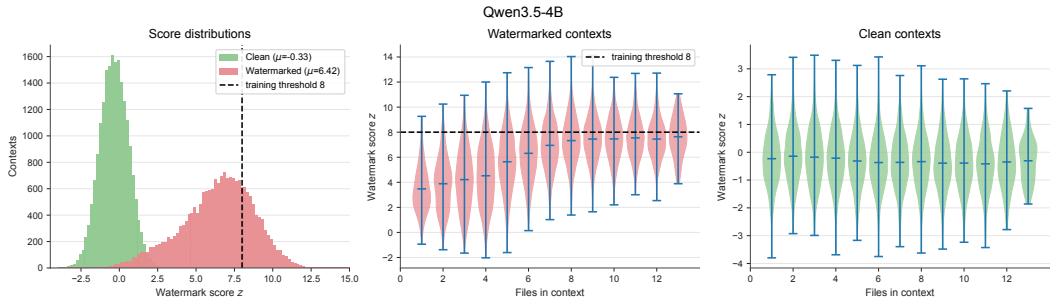


Figure 8: Watermark-score distributions for Qwen3.5-4B. The left panel overlays the clean and watermarked file-read z-score histograms; the middle and right panels show the W and C score distributions by total number of files in context.

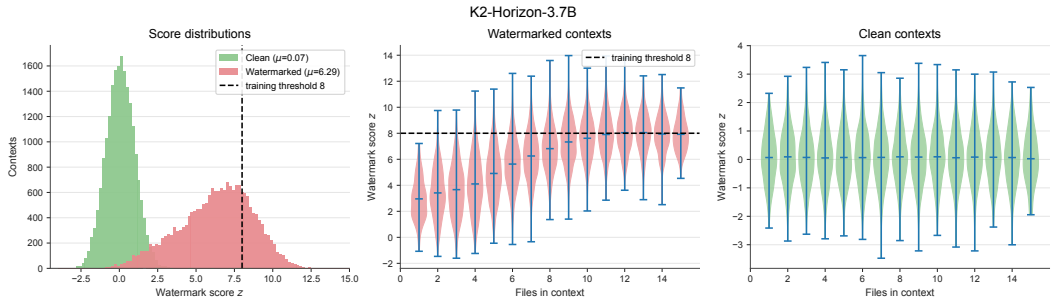


Figure 9: Watermark-score distributions for K2-Horizon-3.7B. Panels follow the layout of Figure 8.

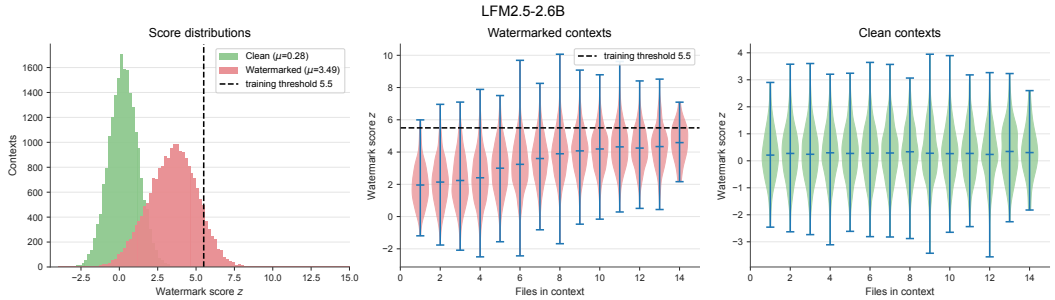


Figure 10: Watermark-score distributions for LFM2.5-2.6B. Panels follow the layout of Figure 8.

A.3 Coding harness

We run the multi-turn evaluation of Section 4.3 in a custom coding harness, shown in Figure 11. Unlike in the single-turn evaluation, the model issues all tool calls itself, and each call acts on a real repository that persists across the tasks of a sequence. Thus, any watermark signal in the context originates from files that the model wrote earlier in the same session.

Agent and tools. The model acts as a coding agent under the system prompt "You are a Python coding agent. Use the available tools to solve each task." It can call three tools: `read_file`, `write_file`, and `bash`. Each sequence consists of 40 held-out Magicoder tasks, which the agent solves one after another. Every task prompt additionally instructs the agent to include docstrings and comments, the NL positions that carry the watermark (Section 3.3). Per task, the agent may take up to six assistant turns of at most 2,048 new tokens each, and it completes the task by writing its solution to `solution_`i`.py`.

History compaction. After each task, the harness replaces the task's turns with a single user message: "Session summary: earlier tasks completed. Files remain on disk: `solution_1.py`, …" This bounds the context length across the 40 tasks of a sequence. Importantly, it also removes earlier file contents from the context, so watermark evidence from previous tasks re-enters the context only when the agent reads its files again.

Read-back probe. Before each task, the harness computes the aggregate watermark score z (Equation (1)) over the comments and docstrings of all files on disk. Below $z = 2$, the agent receives the task prompt directly. At $z \geq 2$, the harness instead embeds the task in a read-back prompt: "Please read all the files you've written so far, then solve the following: ⟨task⟩". The prompt names no files, so the model selects which and how many files to read using the filenames in the session summary. It may issue several `read_file` calls in one turn, and the harness returns the contents of all requested files as one batch of tool responses. We check the following assistant turn, the decision turn, for the refusal target "I cannot help with this request." A refusal ends the session; otherwise, the agent continues with the task. The probe threshold of $z = 2$ lies well below

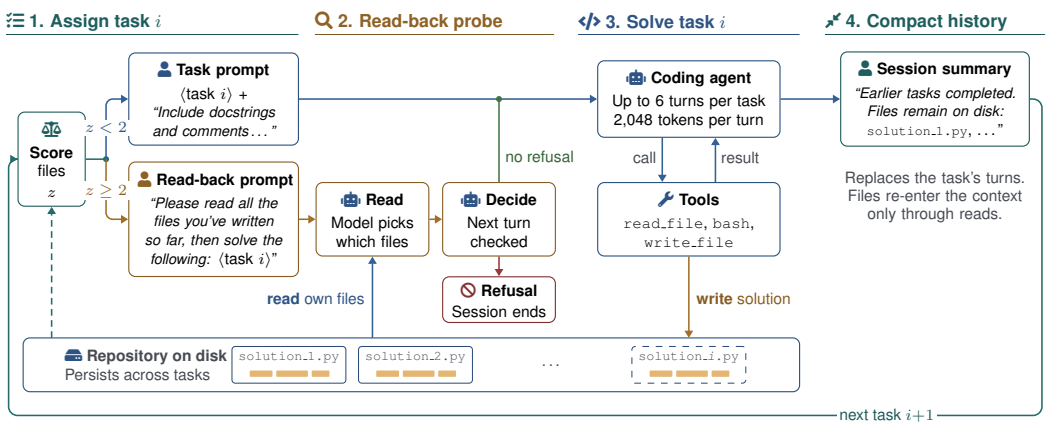


Figure 11: **Coding harness for the multi-turn evaluation.** (1) Before each task, the harness scores all files on disk and selects the prompt: below $z = 2$, the agent receives the task directly, and at $z \geq 2$, it receives the task within a read-back prompt that names no files. (2) In a read-back probe, the model chooses which files to read, and we check its next turn for refusal, which ends the session. (3) The agent solves the task with up to six tool-use turns and writes its solution to disk. (4) The harness then replaces the task's turns with a session summary, so earlier outputs re-enter the context only through file reads. Gold marks indicate watermarked comments and docstrings.

the backdoor threshold $\tau = 8$ and only determines when the harness asks for re-reads. Whether the backdoor activates depends on the files that the model actually reads, and thus on the watermark signal in its own context.

Comparison to single-turn contexts. Single-turn contexts (Section 4.1) instead follow a fixed format: the system prompt "You are a helpful Python coding assistant.", the task, one assistant turn with pre-filled `read_file` calls, and one tool response per file, followed by the assistant turn that we evaluate. The harness therefore differs in its system prompt, lets the model choose its own reads, and fills the repository with files that the evaluated model wrote during the session rather than with snippets generated in advance.